SCHMIB: Segregating Clusters Hierarchically Making Improved Bounds *

John Brevik, Daniel Nurmi, and Rich Wolski†
Computer Science Department
University of California, Santa Barbara
Santa Barbara, California 93106
UCSB Technical Report Number CS2005-27
November, 2006

Abstract

Most space-sharing parallel computers presently operated by high-performance computing centers use batch-queuing systems to manage processor allocation. In many cases, users wishing to use these batch-queued resources have the option of choosing between different queues (having different charging rates) potentially on a number of different machines where they have access. In such a situation, the amount of time a user's job will wait in any one batch queue can significantly impact the overall time a user waits from job submission to job completion. It thus becomes desirable to provide a prediction for the amount of time a job can expect to wait in the queue at a given time. Further, it is natural to expect that attributes of an incoming job, specifically the number of processors requested and the amount of time requested, might impact that job's wait time.

Previous work has shown that it is possible to determine meaningful upper-bounds on queuing delay using a simple non-parametric technique, particularly when site administrators provide information for how jobs should be grouped by processor count.

In this work, we explore the possibility of generating more accurate predictions by automatically grouping jobs having similar attributes using model-based clustering. Moreover, we implement this clustering technique for a time series of jobs so that predictions of future wait times can be generated in real time.

Using trace-based simulation on data from 7 machines over a 9-year period from across the country, comprising over one million job records, we show that clustering either by requested time or by requested number of processors generally produces more accurate predictions than the earlier more naive approaches, that automatic clustering outperforms administrator-determined clusterings, and that clustering by requested time or the product of requested nodes and requested execution time is substantially more effective than clustering by requested number of processors.

*We have chosen a somewhat obscure title for this work in an effort to improve the anonymity of our submission. We will retitle the paper appropriately should it be accepted.

†This work was supported by grants from the National Science Foundation numbered CCF-0331654 and NGS-0305390.

1. Introduction

Typically, high-performance multi-processor compute resources are managed using *space sharing*, a scheduling strategy in which each program is allocated a dedicated set of processors for the duration of its execution. In production computing settings, users prefer space sharing to time sharing, since dedicated processor access isolates program execution performance from the effects of a competitive load. Because processes within a partition do not compete for CPU or memory resources, they avoid the cache and translation look-aside buffer (TLB) pollution effects that time slicing can induce. Additionally, inter-process communication occurs with minimal overhead, since a receiving process can never be preempted by a competing program.

For similar reasons, resource owners and administrators prefer space sharing as well. As long as the time to allocate partitions to, and reclaim partitions from, parallel programs is small, no compute cycles are lost to time-sharing overheads, and resources are efficiently utilized. Thus, at present, almost all production high-performance computing (HPC) installations use some form of space sharing to manage their multi-processor and cluster machines.

Because each program in a space-shared environment runs in its own dedicated partition of the target machine, a program cannot be initiated until there are a sufficient number of processors available for it to use. When a program must wait before it can be initiated, it is queued as a "job" [1] along with a description of any parameters and environmental inputs (*e.g.* input files, shell environment variables, *etc.*) it will require to run. However, because of the need both to assign different priorities to users and to improve the overall efficiency of the resource, most installations do not use a simple first-come-first-served (FCFS) queuing discipline to manage the queue of waiting jobs. Indeed, a number of queue management systems, including PBS [21], LoadLeveler [1], EASY [17], NQS/NQE [20], Maui [19] and GridEngine [13] each offers a rich and sophisticated set of configuration options that allow system administrators to implement highly customized prior-

[1]We will use the term "job" throughout this paper to refer to a description of a program and its execution requirements that a queuing system can use to initiate a program once the necessary resource become available.

ity mechanisms.

Unfortunately, while these mechanisms can be used to balance the need for high job throughput (in order to ensure machine efficiency) with the desires of end-users for rapid turnaround times, the interaction between offered workload and local queuing discipline makes the amount of time a given job will wait highly variable and difficult to predict. Users may wait a long time – considerably longer the the job's eventual execution time – for a job to begin executing. Many users often find this potential for unpredictable queuing delay particularly frustrating since, in production settings, they *can* make fairly reliable predictions of how long a program will execute once it starts running. Without an ability to predict its queue waiting time, however, users cannot plan reliably to have results by a specific point in time.

In this paper, we present a method for automatically predicting bounds, with quantitative confidence levels, on the amount of time an individual job will wait in queue before it is initiated for execution on a production "batch scheduled" resource. The method consists of three interacting but essentially independent components: a percentile estimator, a change-point detector, and a clustering procedure. At a high level, clustering is used to identify jobs of similar characteristics. Within each cluster, job submissions are treated as a time series and the change-point detector delineates regions of stationarity. Finally, the percentile estimator computes a quantile that serves as a bound on future wait time based only on history from the most recent stationary region in each cluster. All three components can be implemented efficiently so that online, real-time predictions are possible. Thus, for each job submission, our method can generate a predicted bound on its delay using a stationary history of previous jobs having similar quantitative characteristics. In addition, as jobs complete their time in queue, new data becomes available. Our method automatically incorporates this information by adjusting its clustering and change-point estimates.

In previous work [26, 26] we have investigated various methods for percentile estimation and explored the efficacy of automatic change-point detection in such highly correlated data. Using a simple method based on binomial distributions, combined with on-line autocorrelation analysis, we have found that it is possible to predict bounds on the delay of individual jobs that are tighter then parametric methods based on Maximum Likelihood Estimation (MLE) of Weibull and log-normal distributions. This methodology we term the *Binomial Method Batch Predictor* (BMBP).

In this work, we focus on improving the bounds achieved by BMBP through a novel approach to model-based clustering applied hierarchically to the job submission history it uses. We describe the full methodology (Binomial percentile estimator, change-point detector, and clustering algorithm) and discuss the often complicated interaction between change-point detection and re-clustering in response to newly available queue delay measurements when the system is to be used in an on-line setting. Thus, our goal is to explore the effect of **s**egregating **c**lusters **hi**erarchically to **m**ake **i**mproved **b**ounds (abbreviated as SCHMIB).

To verify the effectiveness of the approach, we compare BMBP with and without SCHMIB using job submission traces from 7 supercomputers (including three currently in operation) operated by the National Science Foundation and the Department of Energy over the past 9 years comprising approximately one million job submissions. By examining job arrival time, requested execution time, and requested node count, we simulate each queue in each trace and compute a prediction for each job. Our results indicate that BMBP (which is more effective than competitive parametric methods) used with SCHMIB achieves significantly tighter bounds on job wait time in most cases. Thus, this new combination represents the most accurate available method for predicting bounds on individual job delay times, and does so with a specifiable degree of certainty.

Thus, this paper makes three significant new contributions with regard to predicting individual job queue delays.

- We present BMBP (briefly) and SCHMIB as an example of an accurate, non-parametric, and fully automatic method for predicting bounds (with specific levels of certainty) on the amount of queue delay each individual job will experience.
- We verify the efficacy of these techniques using job submission logs from currently operating large-scale batch systems, and from archival logs for systems that are no longer in operation.
- We find that SCHMIB improves the accuracy of the bounds, that requested execution time is a more significant factor for clustering jobs than is processor count that jobs cluster differently from the way that expert site administrators (who control the specific scheduling policies) have anticipated.

We believe that these results constitute a new and important capability for users of batch-controlled resources. Using an online, web-based, real-time version of BMBP and SCHMIB [26] that allows users to generate predictions on demand, these users are better able to decide on which machines to use, which queues on those machines to use, the maximum amount of run time to request, and the number of processors to request so as to minimize job turnaround time or maximize the utilization of their respective time allocations. Our techniques are also useful as a scheduling policy diagnostic for site administrators. For example, our results indicate that the amount of requested execution time is a far more significant factor in determining queue delay than is requested processor count (presumably due to back-filling [16]). One site administrator at a large scale computer center [2] expressed surprise at this result, since she believed she had set the scheduling policy at this site to favor jobs with large processor counts in an effort to encourage users to use the resource for "big" jobs. Because short jobs can be more readily scheduled when back-filling is used, users are circumventing the site policy and submitting small jobs to improve turn-around time. This example illustrates how prototype versions of BMBP and SCHMIB are already having an impact in large-scale batch-controlled settings. We discuss the nature of this impact further in Section 4.

This ability to make predictions for individual jobs distinguishes our work from other previous efforts. An extensive body of re-

[2]The specific administrator and site have been elided to improve the anonymity of our blind submission but will be included in a final version of the paper, should it be accepted.

search [3, 5, 6, 8, 9, 10, 11, 24] investigates the statistical properties of offered job workload for various HPC systems. In most of these efforts, the goal is to formulate a *model* of workload and/or scheduling policy and then to derive the resulting statistical properties associated with queuing delay through simulation. Our approach focuses strictly on the problem of *forecasting* future delay bounds; we do not claim to offer an explanatory, or even a descriptive, model of user, job, and/or system behavior. However, perhaps because of our narrower focus, our work is able to achieve predictions which are, in a very specific and quantifiable sense, more accurate and more meaningful than those reported in the previous literature. We discuss related approaches further in Section 2.

In the next section, we describe BMBP briefly and SCHMIB in detail. As mentioned previously, Section 4 discusses our evaluation procedure and the specific results we have achieved, Section 2 covers related approaches and efforts, and finally in Sectionsec:conclusions we recap and conclude our description of this work.

2. Related Work

Previous work in this field can be categorized into three groups. The first group of work belongs under the general heading of scheduling jobs on parallel supercomputers. In a works by Feitelson and Rudolph [9, 10], the authors outline various scheduling techniques employed by different supercomputer architectures and point out strengths and deficiencies of each. The prevalence of distributed memory clusters as supercomputer architectures has led to most large scale sites using a form of "variable partitioning" as described in [9]. In this scheme, machines are space shared and jobs are scheduled based on how many processors the user requests and how much time they specify as part of the job submission. As the authors point out, this scheme is effective for cluster type architectures, but leads to fragmentation as well as potentially long wait times for jobs in the queue.

The second field of previous work which is relevant to our work involves using various models of large scale parallel job scenarios to predict the amount of time jobs spend waiting in scheduler queues. These works attempt to show that batch queue job wait times can be inferred under the conditions that one knows the length of time jobs actually execute and that the algorithm employed by the scheduler is known. If both conditions are met, it has been shown that the mean job wait time can be predicted, as shown in a paper from Smith, Taylor and Foster [24], but even when the job execution times well modelled, the mean error ranges from 33 to 73 percent. In this work, the authors use a template-based approach to categorize and then predict job execution times. From these execution-time predictions, they then derive mean queue delay predictions by simulating the future behavior of the batch scheduler in faster-than-real time. Downey [5, 6] uses a similar set of assumptions for estimating queue wait times. In this work, he explores using a log-uniform distribution to model the remaining lifetimes of jobs executing in all machine partitions as a way of predicting when a "cluster" of a given size will become available and thus when the job waiting at the head of the queue will start. As a metric of success, Downey uses the correlation between the wait time of the head job, if execution times are estimated using his model, and the head job wait time if the execution time is exactly known. Both of these approaches make the underlying assumption that the scheduler is employing a fairly straightforward scheduling algorithm (one which does not allow for special users or job queues with higher or lower priorities), and also that the resource pool is static for the duration of their experiments (no downtimes, administrator interference, or resource pool dynamism).

Our work differs from these approaches in two significant ways. First, instead of inferring from a job execution model the amount of time jobs will wait, we make job wait time inference from the actual job wait time data itself. The motivation for why this is desirable stems from research efforts [4, 14], which suggest that modelling job execution time may be difficult for large-scale production computing centers. Further, making inference straight from the job wait time data, we avoid having to make underlying assumptions about scheduler algorithms or machine stability. We feel that in a real world scenario, where site scheduling algorithms are rarely published and are not typically simple enough to model with a straightforward procedure, it is unlikely that valid queue wait time predictions can be made with these assumptions.

Secondly, our approach differs in the statistic we use as a prediction. Most often, we see researchers using as a prediction the mean amount of time a job is expected to wait in the queue. Our approach instead uses bounds on the time an individual job will wait rather than a specific, single-valued prediction of its waiting time. We contend that the highly variable nature of observed queue delay is better represented to potential system users as quantified confidence bounds than as a specific prediction, since users can "know" the odds that their job will fall outside the range.

3. BMBP and SCHMIB

In this section, we describe our approach to the three related problems that we must solve to implement an effective predictor: quantile estimation[3], change-point detection, and clustering. The general approach we advocate is first to cluster the observed job submission history according to jobs having similar quantitative characteristics (e.g. requested node count, requested maximum execution time, or requested node-hours), next to identify the most recent region of stationarity in each cluster (treated as a time series), and finally to estimate a specific quantile from that region to use as a statistical bound on the time a specific job will wait in queue. While logically the steps occur in this order, we describe them in reverse order, providing only a summarization of our quantile estimation and stationarity approaches, primarily due to space constraints but also because we have analyzed these extensively in other publications [26, 26].

3.1 Inference for Quantiles using The Binomial Method

Our goal, with this method, is to determine an upper bound on a specific quantile at a fixed level of confidence, for a given population whose distribution is unknown. If the quantile were known with certainty, and the population were the one from which a given job's queue delay were to be drawn, this quantile would

[3]We use the term "quantile" instead of the term "percentile" throughout the remainder of this paper.

serve as a statistical bound on the job's waiting time. For example, the 0.95 quantile for the population will be greater than or equal to the delay experienced by all but 5% of the jobs. Colloquially, it can be said that the job has a "95% chance" of experiencing a delay that is less than the 0.95 quantile. We assume that the quantile of interest (0.95, 0.99, 0.50, etc.) is supplied to the method as a parameter by the site administrator depending on how conservative she believes the estimates need to be for a given user community.

However, since the quantiles cannot be known exactly and must be estimated, we use an upper confidence bound *on the quantile* that, in turn, serves as a conservative bound on the amount of delay that will be experienced by a job. To be precise, to say that a method produces an upper 95% confidence bound on the a given quantile implies that the bound produced by this method will, over the long run, overestimate the true quantile 95% of the time. The degree of conservatism we assume is also supplied to the method as a confidence level. In practice, we find that while administrators do have opinions about what quantile to estimate, the confidence level for the upper bound is less meaningful to them. As a result, we typically recommend estimating what ever quantile is desired by the upper 95% confidence bound on that quantile.

Our approach, which we term the *Binomial Method*, is based on the following simple observation: If X is a random variable, and X_q is the q quantile of the distribution of X, then a single observation x from X will be greater than X_q with probability $(1-q)$. (For our application, if we regard the wait time, in seconds, of a particular job submitted to a queue as a random variable X, the probability that it will wait for less than $X_{.95}$ seconds is exactly $.95$.)

Thus (provisionally under the typical assumptions of independence and identical distribution) we can regard all of the observations as a sequence of independent Bernoulli trials with probability of success equal to q, where an observation is regarded as a "success" if it is less than X_q. If there are n observations, the probability of exactly k "successes" is described by a Binomial distribution with parameters q and n. Therefore, the probability that more than k observations are greater than X_q is equal to

$$1 - \sum_{j=0}^{k} \binom{n}{j} \cdot (1-q)^j \cdot q^{n-j} \qquad (1)$$

Now, if we find the smallest value of k for which Equation 1 is larger than some specified confidence level C, then we can assert that we are confident at level C that the k^{th} value in a sorted set of n observations will be greater than or equal to the X_q quantile of the underlying population – in other words, the k^{th} sorted value provides an *upper level-C confidence bound* for X_q.

Clearly, as a practical matter, neither the assumption of independence nor that of identical distribution (stationarity as a time series) holds true for observed sequences of job wait times from the real systems, and these failures present distinct potential difficulties for our method.

Let us first (briefly) address the issue of independence, assuming for the moment that our series is stationary but that there may be some autocorrelation structure in the data. We hypothesize that the time-series process associated to our data is *ergodic*, which roughly amounts to saying that all the salient sample statistics asymptotically approach the corresponding population parameters. Ergodicity is a typical and standard assumption for real-world data sets; *cf., e.g.,*[12]. Under this hypothesis, a given sample-based method of inference will, *in the long run*, provide accurate confidence bounds.

Although our method is not invalidated by dependence, a separate issue from the *validity* of our method is that exploiting any autocorrelation structure in the time series should, *in principle*, produce more accurate predictions than a static binomial method which ignores these effects. Indeed, most time-series analysis and modeling techniques are primarily focused on using dependence between measurements to improve forecasting [2]. For the present application, however, there are a number of obfuscating factors that foil typical time-series methods. First of all, for a given job entering a queue, there are typically several jobs in the queue, so that the most recent available wait-time measurement is for several time-lags ahead. The correlation between the most recent measurement at the time a job enters the queue and that job's eventual wait time is typically modest, around 0.1, and does not reliably contribute to the accuracy of wait-time predictions. Another issue is the complexity of the underlying distribution of wait times: They typically have more weight in their tails than exponential distributions, and many queues exhibit bimodal or multimodal tendencies as well. All of this makes any linear analysis of data relationships (which is the basis of the "classical" time-series approach) very difficult. Thus while the data is not independent, it is also not amenable to standard time-series approaches for exploiting correlation.

3.2 Correct and Accurate Predictions

Because we are predicting a probabilistic bound on the delay for each job, it is useful to differentiate between a correct prediction and an accurate one in this context. We define a *correct* prediction to be one that is greater than or equal to a job's eventual queueing delay, and a *correct predictor* to be one for which the total fraction of correct predictions is greater than or equal to the success probability specified by the target quantile. For example, a correct predictor of the 0.95 quantile generates correct predictions for at least 95% of the jobs that are submitted.

Notice that it is trivial to specify a correct predictor under this definition. For example, to achieve a correct prediction percentage of 95%, a predictor could return an excessively large prediction (*e.g.,* a predicted delay of several years) for 19 of every 20 jobs, and a prediction of 0 for the 20^{th}. To distinguish among correct predictors, we compare their *accuracy* in terms of the error they generate, where error is some measure of the difference between predicted value and the value it predicts.

In this work, we will use Root Mean Square (RMS) error for the over-predictions as a measure of accuracy for correct predictors. We consider only over-prediction error, as we believe that the error generated for the percentage of jobs that are incorrectly predicted is relatively unimportant to the user. For example, among predictors that are 95% correct, it is our contention that users would prefer one that achieves lower over-prediction error for the

95% of the jobs it predicts correctly over one that achieves a lower error rate on the 5% that are incorrectly predicted at the expense of greater overall error in the correct predictions.

Note that comparing predictors strictly in terms of their error (without consideration of their correctness) is difficult. For example, a predictor that estimates the mean of each stationary region will generate a lower RMS than one that estimates the 0.95 quantile, but the mean predictor will not provide the user with a meaningful delay bound (*i.e.,* one having a probability value attached to it). Thus, for a given job workload, we only compare predictor accuracy among those predictors that are correct.

Note also that, while RMS error is used widely as a measure of accuracy for predictions of expected values (*e.g.* in time series), its meaning is less clear in the context of quantile prediction. In this paper, we are focusing on estimating a time value which is greater than the wait time of a specific job with probability .95. Therefore, if the distribution of wait times is highly right-skewed, a predictor may be working quite well and still have a very high RMS error. Thus, the actual *value* of the RMS error is not particularly meaningful; however, it is still useful as a means of *comparison*: For a particular set of jobs, if one correct prediction method has a lower RMS than another, then that first method is preferable in terms of producing tighter, less conservative upper bounds (*cf.* Section 4.5 for further discussion of RMS error).

3.3 Non-stationarity and Change-Point Analysis

Unlike the issue of independence and correlation, the issue of non-stationarity *does* place limitations on the applicability of our method. Clearly, for example, it will fail in the face of data with a "trend," say, a mean value that increases linearly with time. On the other hand, insisting that the data be stationary is too restrictive to be realistic: Large compute centers change their scheduling policies to meet new demands, new user communities migrate to or from a particular machine, *etc.* It seems to be generally true across the spectrum of traces we have examined that wait-time data is typically stationary for a relatively long period and then undergoes a "change-point" into another stationary regime with different population characteristics. We thus use the Binomial Method as a prediction method for data which are stationary for periods and for which the underlying distribution changes suddenly and relatively infrequently; we next discuss the problem of detecting change-points in this setting.

Given an independent sequence of data from a random variable X, we deem that the occurrence of three values in a row above $X_{.95}$ constitutes a "rare event" and one which should be taken to signify a change-point. Why three in a row? To borrow a well-known expression from Tukey [4], two is not enough and four is too many; this comes from consideration of "Type I" error. Under the hypothesis of identical distribution, a string of two consecutive high or low values occurs every 400 values in a time series, which is an unacceptable frequency for false positives. Three in a row will occur every 8000 values; this strikes a balance between sensitivity to a change in the underlying distribution of the population and certainty that a change is not being falsely reported.

[4]We refer here to Tukey's notorious explanation why the "whiskers" in a boxplot should extend 1.5 IQRs, namely that "1 is too small and 2 is too large"; beyond its beautiful "sound bite" quality, Tukey's quote serves as a reminder that any statistical threshold, such as 95% confidence or .05 significance level, is an artificial entity ultimately chosen for its usefulness.

Now, suppose that the data, regarded as a time series, exhibits some autocorrelation structure. If the lag-1 autocorrelation is fairly strong, three or even five measurements in a row above the .95 quantile might not be such a rare occurrence, since, for example, one unusually high value makes it more likely that the next value will also be high. In order to determine the number of consecutive high values (top 5% of the population) that constitute a "rare event" approximately in line with the criterion spelled out for independent sequences, we conducted a Monte Carlo simulation with various levels of lag-1 autocorrelation in $AR(1)$ time series [12], observed the frequencies of occurrences of consecutive high and low values, and generated a lookup table for rare-event thresholds. Thus, to determine if a change-point has occurred, we compute the autocorrelation of the most recent history, look up the maximum number of "rare" events that should normally occur with this level of autocorrelation, and determine whether we have surpassed this number. If so, our method assumes the underlying system has changed, and that the relevant history must be trimmed as much as possible to maximize the possibility that this history corresponds to a region of stationarity. Note that indiscriminate history-trimming will not allow our method to function properly, since the resulting small sample sizes will generate unnecessarily conservative confidence bounds.

The minimum useful history length depends on the quantile being estimated and the level of confidence specified for the estimate. For example, it follows from Equation 1 above that in order to produce an upper 95% confidence bound for the .95 quantile, the minimum history size that can be used is 59. (This reflects the fact that $.95^{59} < .05$, while $.95^{58} > .05$.)

Again, a more complete description of the Binomial Method and its concomitant procedure for change-point detection, as well as a more detailed rationale and analysis of the assumptions upon which it is based are available in [26], as is evidence of its effectiveness in comparison to other methods. Additional evidence for its effectiveness is described in [26]. Here we endeavor only to summarize the approaches, which we will heretofore refer to together as the **B**inomial **M**ethod **B**atch **P**redictor (BMBP), and provide a general motivation for their effectiveness.

3.4 SCHMIB: Prediction with Model-Based Clustering

According to our observations and to anecdotal evidence provided by users and site administrators, there are differences among the wait times various jobs might expect to experience in the same queue, based purely on characteristics of the jobs such as the amount of time and the number of nodes requested. This is certainly easy to believe on an intuitive level; for example, if a particular queue employs backfilling [16], it is more likely that a shorter-running job requesting a smaller number of nodes will be processed during a time when the machine is being "drained." Thus, for a given job, we might hope to make a better prediction for its wait time if we took its characteristics into account rather than making one uniform prediction which ignores these characteristics.

On the other hand, the same difficulties arise in trying to produce regression models [24] as we encountered in the problem of trying to use autoregressive methods: In particular, the data are typically multimodal and do not admit of simple parametric models. We therefore explore the idea of *clustering* the data into groups having similar attributes, so that we can use our non-parametric predictor on each cluster separately.

In fact, in [26], based on advice we received from several expert site administrators for currently operating systems, we employed a rather arbitrary partitioning of jobs in each queue by processor count, running separate predictors within each partition, which resulted in substantially better predictions. However, it would clearly be desirable to find a partition which is in some (statistical) sense "optimal" rather than relying on such arbitrary methods; for our purposes, it is also desirable to find a partitioning method that can be machine-learned and is therefore applicable across different queues with different policies and user characteristics without direct administrator intervention or tuning. Moreover, as a diagnostic tool, it would be advantageous to be able to compare the machine-determined clustering with that determined by site administrators to illuminate the effects of administrator-imposed scheduling policies. In this section, we describe our approach to this problem, which falls under the rubric of *model-based clustering* [15, 22, 25].

3.5 Model-Based Clustering

The problem of partitioning a heterogeneous data set into clusters is fairly old and well studied [15, 18, 22, 25]. The simplest and most common clustering problems involve using the values of the data, relative to some notion of distance. Often, one postulates that the distribution within each cluster is Gaussian, and the clusters are formed using some well-known method, such as the so-called k-means algorithm [18] or one of various "hierarchical" or "partitional" methods [22, 25]. If the number of clusters is also unknown, a model-selection criterion such as BIC [23], which we will discuss further below, is often used to balance goodness of fit with model complexity.

In fact, it is tempting, if for no other reason than that of simplicity, to form our clusters in this way, according to how they naturally group in terms of one or more job attributes. Note, however, that this method of clustering in no way takes into account the wait times experienced by jobs, which is ultimately the variable of interest; it is by no means clear that a clustering of jobs by how their requested wait times group will result in clusters whose wait-time distributions are relatively homogeneous. For example, it is possible that a subset of the requested job execution times form a nice Gaussian cluster between 8 and 12 minutes, but that due to some combination of administrative policy, backfilling, and various "random" characteristics of the system as a whole, jobs requesting less than 10 minutes experience substantially different wait times than those requesting more than 10 minutes, so this cluster is actually meaningless in terms of predicting wait times.

In our case, then, the situation is somewhat more complicated than ordinary clustering: We wish to cluster the data according to some characteristics which are *observable at the time the job is submitted* (explanatory variables), but using the actual wait times (response variable) as the basis for clustering. That is, we wish to use observed wait times to cluster jobs, but then to determine how each cluster is characterized by quantitative attributes that are available when each job is submitted so that an arriving job can be categorized before it begins to wait. In the discussion that follows, we will use the *requested execution time* (used to implement backfilling) as the explanatory characteristic, but this is only for the sake of ease of exposition.

The idea behind our method runs as follows: We postulate that the set of requested times can be partitioned into k clusters $C_1, \ldots, C_k$, which take the form of intervals on the positive time axis, such that within each C_j the wait times are governed by an exponential distribution with unspecified parameter λ_j.

The choice of exponential distributions is something of an oversimplification – in fact a Weibull, log-normal or hyperexponential would probably be a more accurate choice – but the fact that the clusters are relatively homogeneous makes the exponential model accurate enough with relatively little computational expense; moreover, in practice, exponentials are more than discerning enough to produce an adequate number of clusters. As a check, we generated an artificial trace using different log-normally distributed wait times corresponding to the intervals of requested times $[1, 100]$, $[101, 200]$, $[201, 300]$, $[301, 400]$, and $[401, 500]$ and fed this data to our clustering method. It recovered the following clusters for the data: $[1, 39]$, $[40, 40]$, $[41, 100]$, $[101, 197]$, $[198, 300]$, $[301, 398]$, $[399, 492]$, $[493, 493]$, $[494, 500]$. Since our method always clusters the ends together to ensure that these clusters contain at least 59 elements, the exponential clustering method recovers the original clusters almost exactly.

We assume that the appropriate clustering is into connected intervals along the time axis; this provides an intuitive model for the eventual users of our predictions. Given a desired value for the number k of clusters, then, we use a modified form of *hierarchical clustering*. According to this method, we start with each unique value for the requested time in its own cluster. We then merge the two adjacent (in the sense of adjacency on the time axis) clusters that give the largest value of the *log-likelihood function* $\log L$, calculated jointly across the clusters, according to the maximum-likelihood estimators for the exponential parameters λ_j, which are given by $\frac{\#(C_j)}{\sum_{x \in C_j} x}$. This process continues until the number of clusters is equal to k. Note that this is a well-accepted method for clustering [18, 22, 25]; however, it does not guarantee that the resulting clustering will maximize the log-likelihood over all possible choices of k clusters, even if we assume that the clusters are all intervals. This latter problem is prohibitively expensive computationally for an on-line, real-time application, even for moderately large data sets, and we are therefore forced to use some restricted method.

Each arriving job can then be categorized by identifying the cluster whose minimum and maximum requested time straddle the job's requested time.

Continuing, the question of which value of k to use is a problem in *model selection*, which recognizes the balance between modeling data accurately and model simplicity. The most generally accepted model-selection criterion is the *Bayes Information*

Criterion (BIC) [23], the form of which is

$$\mathrm{BIC}(\theta) = \log L(\theta) - \frac{p}{2}\log n,$$

where θ stands for the (vector of) free parameters in the model, L is the joint likelihood function across the whole data set, calculated using the MLE for θ, p is the dimensionality of θ ($2k-1$ in our case: the $k-1$ break points on the time axis to define our clusters, and the k values for the λ_j, all of which are scalars), and n is the total sample size. The first term in the BIC formula should be seen as a measure of goodness of fit, while the second term is a "penalty" for model complexity (*i.e.* one with a large number of parameters). It is always true that for a less restricted model (in our case, one allowing a larger number of clusters), the $\log L$ term will be larger, so the penalty function is critical to avoid over-parameterizing. Maximizing the BIC expression over a set of proposed models has good theoretical properties and generally produces good results in practice. Thus, our clustering strategy is to specify a range of acceptable k-values; perform the hierarchical clustering described above for each of these values of k; and then calculate the BIC expression for each resulting clustering and choose the one for which BIC is greatest.

3.6 Change-point Detection and Clustering

There is a potential difficulty implementing re-clustering as new job delay information becomes available. When SCHMIB finds a new clustering, it does not take into account the minimum necessary history required by BMBP (calculated, using the quantile of interest and desired confidence level, from Equation 1). As a result, clusters determined by SCHMIB may not be suitable for use by BMBP. We address this problem in the following way. When BMBP is required to make a forecast from a cluster that does not contain enough data, it augments the history it considers by adding the data from the next *higher* cluster in the adjacency list, adding histories from further higher clusters if necessary until enough history is available. This temporary merge is recomputed each time a forecast from a short cluster is needed, so that it is only done for a given cluster until that cluster has enough data. We term this process *history borrowing.* Borrowing is done from *higher* adjacent clusters in the interest of safeguarding the correctness of the predictor, possibly at the expense of generating slightly over-conservative estimates.

4. Results

In this section, we describe our method for evaluating BMBP and SCHMIB, and we then detail a set of simulation experiments that use, as their input, traces of job submission logs gathered at various supercomputing centers. While we have implemented and deployed a prototype of BMBP at a number of nationally accessible large-scale sites, and we are in the processes of enhancing this prototype with SCHMIB, a rigorous comparison is best served by repeatable, trace-based simulation. We describe the details of the simulations (we use a single simulator that can parse each job log) and then report the prediction performance users *would have* seen had BMBP and/or SCHMIB been available at the time each job in each trace was submitted.

We investigate the problem in terms of estimating an upper

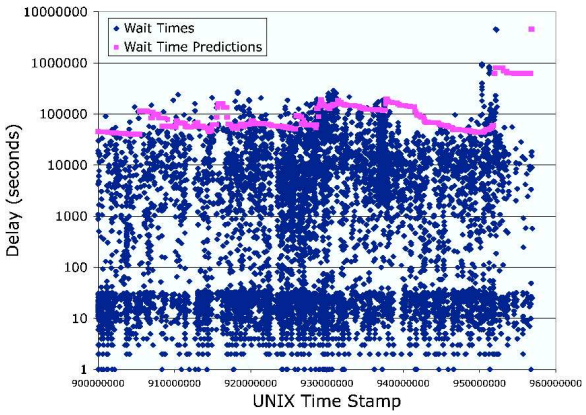

Figure 1. Example output of BMBP simulator without the use of SCHMIB clustering or static grouping.

bound on the 0.95 quantile of queuing delay, however our approach can be similarly formulated to produce lower confidence bounds, or two-sided confidence intervals, at any desired level of confidence. It can also be used, of course, for any population quantile. For example, while we have focused in this paper on the relative certainty provided by the .95 quantile, our method estimates confidence bounds for the median (*i.e.*, the point of "50-50" probability) with equal effectiveness. We note that the quantiles at the tail of the distribution corresponding to rarely occurring but large values are more variable, hence more difficult to estimate, than those nearer the center of the distribution. Thus, in a typical batch queue setting, which is characterized by large numbers of jobs experiencing short wait times and a few jobs experiencing long wait times, the upper quantiles provide the greatest challenge for a prediction method. By focusing on an upper bound for the .95 quantile, we are testing the limits of what can be predicted for queue delay.

Note also that our assertion of retroactive prediction correctness and accuracy assumes that users would not have changed the characteristics of the jobs they submitted in response to the availability of the quantile predictions we generate. Moreover, the online prototype we have developed, while operational, is in use by only a few users [5], making difficult an analysis of whether BMBP and SCHMIB predictions affect workload characteristics. However, unless such feedback induces undamped oscillation resulting in frequent "spikes" in delay (cf. Subsection 4.5 below), BMBP and SCHMIB are likely to continue to make correct and accurate predictions. We do plan to monitor the workloads experienced by various sites after BMBP and SCHMIB are deployed for general use at various large-scale sites and report on the effects as part of our future work.

4.1 Simulation

Our simulator takes as input a file containing historical batch-

[5] We count ourselves in the category of BMBP and SCHMIB users, as we used the on-line system to schedule the simulations we executed for this paper at various sites where our system is currently operating.

queue job wait times from a variety of machines/queue combinations and parameters directing the behavior of our models. For each machine/queue for which we have historical information, we were able to create parsed data files which contain one job entry per line comprising the UNIX time stamp when the job was submitted, the duration of time the job stayed in the queue before executing, the amount of requested execution time, and the node count.

The steady-state operation of the simulation reads in a line from the data file, makes a prediction (using BMBP or BMBP and SCHMIB) and stores the job in a "pending queue". We then increment a virtual clock until one of two events occur:

- The virtual time specified for the job to wait in the pending expires.
- A new job enters the system according to the virtual clock.

When the first case occurs, the job is simply added to a growing list of historical job wait times stored in memory. Although the waiting time for the job is carried in the trace, the predictor is not entitled to "see" the waiting time in the history until it stops waiting in queue and is released for execution. When the historical record changes, BMBP is given the new record so that it can update its internal state. If SCHMIB is in use, the BMBP predictor(s) for the cluster(s) into which the jobs are placed are updated.

When the second case occurs, the current prediction value is used to make a prediction for the job entering the queue, the simulation checks to see if the predicted time for that job is greater than or equal to the actual time the job will spend in the pending queue (success), or the predicted time was less than the actual job wait time (failure). The success or failure is recorded, and the job is placed on the pending queue. Note that in a "live" setting this success or failure could only be determined after the job completed its waiting period.

In addition to events which occur with respect to time, one final simulation event, re-clustering, occurs after a pre-determined number (1000 in our study) of simulated job arrivals have occurred. That is, the simulator presents new delay measurements to BMBP when the simulated waiting times have expired, records forecast accuracy, and triggers re-clustering. The code implementing BMBP and SCHMIB are modularized so that the same implementation can be operated by the simulator or by the operational infrastructure that makes "live" job predictions in real time.

As an example, in Figure 1 we show the time series of job queue delays and 0.95 quantile predictions generated by BMBP without SCHMIB for the SDSC SP-2 machine and the *high* queue. On the y-axis using a log scale we show delay measured in seconds. Along the x-axis are Unix time stamps. Each dark-colored graph feature represents the queue delay experienced by a particular job, and the light-colored features near the top represent the predictions.

In contrast, Figure 2 shows the same data as does Figure 1 in dark graph features, but the 0.95 quantile predictions generated by BMBP and SCHMIB using requested execution time as a response variable as light graph features. Note that although it may appear

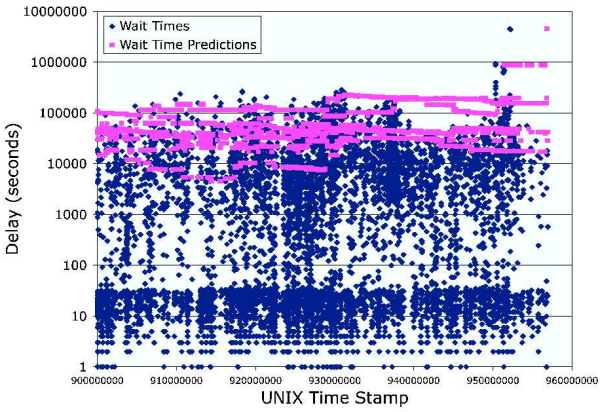

Figure 2. Example output of BMBP/SCHMIB simulator when using the requested time to cluster jobs and make predictions.

as though there are multiple predictions for any given point in time, each job can only belong to one cluster, and therefore the graph is composed of unique job wait time/prediction pairs.

Notice that many of the predictions shown in Figure 2 are almost an order of magnitude lower than the predictions for the corresponding point in time in Figure 1. Thus if the SCHMIB-determined clusters correctly categorize jobs by their response variables, BMBP applied to each cluster should yield correct predictions that are more accurate (produce a "tighter" bounds) than without clustering.

4.2 Data Sets

We obtained 7 archival batch-queue logs from different high-performance production computing settings covering different machine generations and time periods. From each log, we extracted data for the various queues implemented by each site. For all systems except the ASCI Blue Pacific system at Lawrence Livermore National Laboratory (LLNL), each queue determines, in part, the priority of the jobs submitted to it.

The job logs come from three machines operated by the San Diego Supercomputer Center during three different periods: the the IBM SP-2 (**sdsc**), The SDSC "Blue Horizon" (**sdscblue**) and the IBM Power-4 system (**datastar**). We also use traces from the Cornell Theory Center (ctc) Lawrence Livermore National Laboratory's SP-2 (**llnl**), the Cray-Dell cluster operated by the Texas Advanced Computing Center (**lonestar**), and the Argonne National Labs/University of Chicago TeraGrid (**ucteragrid**). The **ctc**, **sdsc**, and **sdscblue** logs we obtained from Feitelson's workload web site [7], the **llnl** data appears courtesy of Brent Gorda at LLNL, and we gathered the **datastar**, **lonestar**, and **ucteragrid** traces using our own infrastructure for real-time predictions. Collectively comprises over one million job submissions spanning approximately an 9-year period. The site and queue names, the durations covered by each job log, and the number of predictions we make for each queue is shown in Table 1. Within each log, each job is repre-

Machine/Queue	Start Date	End Date	Pred Count
ctc/all	Jun 1996	May 1997	76206
datastar/express	Apr 2004	Apr 2005	18242
datastar/high	Apr 2004	Apr 2005	6781
datastar/normal	Apr 2004	Apr 2005	63751
datastar/TGnormal	Apr 2004	Apr 2005	6091
llnl/all	Jan 2002	Oct 2002	54953
lonestar/normal	Jan 2004	Mar 2005	27486
lonestar/serial	Jan 2004	Mar 2005	2181
sdscblue/express	Apr 2000	Dec 2002	66320
sdscblue/high	Apr 2000	Dec 2002	15781
sdscblue/low	Apr 2000	Dec 2002	16104
sdscblue/normal	Apr 2000	Dec 2002	47407
sdsc/express	Apr 1998	Apr 2000	3985
sdsc/high	Apr 1998	Apr 2000	7794
sdsc/low	Apr 1998	Apr 2000	21126
sdsc/normal	Apr 1998	Apr 2000	29765
ucteragrid/dque	Jan 2004	Oct 2005	58163

Table 1. Machine and queue names, start and end dates for each log, and the number of predictions made from each log.

Machine/Queue	plain	static	nodes	rtime	rnprod
ctc/all	96.0	95.8	95.7	96.2	95.9
datastar/express	97.2	96.6	96.5	95.3	95.4
datastar/high	95.1	94.2	93.4	92.0	92.1
datastar/normal	95.8	95.8	95.5	95.5	94.7
datastar/TGnormal	97.3	97.3	97.3	94.6	94.7
llnl/all	96.4	96.9	96.3	96.9	96.7
lonestar/normal	95.3	95.7	95.9	97.4	97.2
lonestar/serial	96.2	96.2	96.2	95.0	95.0
sdscblue/express	93.6	93.0	93.3	95.7	96.1
sdscblue/high	96.1	95.6	95.1	95.1	94.7
sdscblue/low	95.0	94.8	94.5	94.8	93.2
sdscblue/normal	96.0	95.8	95.3	95.2	95.2
sdsc/express	96.7	96.4	96.7	96.5	96.2
sdsc/high	96.5	96.3	95.5	95.7	95.9
sdsc/low	95.5	95.0	94.6	94.6	94.5
sdsc/normal	96.5	96.3	95.2	95.5	95.1
ucteragrid/dque	97.0	97.0	97.0	96.8	96.8

Table 2. Correctness percentages for five BMBP simulation methods for predicting the .95 quantile with 95% confidence.

sented uniquely by four values: submission time, queue wait time, number of nodes requested and number of seconds requested.

In addition, we contacted several site administrators, who have graciously allowed us to install and experiment with the BMBP real-time prediction system. After many consultations, the administrators furnished us with a scheme for node partitioning in the currently active systems that they believe would improve BMBP. That is, we asked each administrator to give us a static node clustering to use for his or her site. After several iterations, we converged on a single static clustering of 1 to 4 nodes, 5 to 16 nodes, 17 to 64 nodes, and greater than or equal to 65 nodes, which we currently use for the real-time prediction system.

4.3 Nodes, Runtime, and Node-Seconds

In the following two tables we summarize the performance of BMBP and SCHMIB. Each table compares the performance of "plain" BMBP (without SCHMIB), BMBP applied to the "static" node clustering obtained from expert administrators, and the combination of BMBP and SCHMIB when clustering by requested "nodes", maximum execution time, abbreviated "rtime", and the product of the two (node-seconds) respectively, abbreviated "rnprod."

In Table 2 we show the name of each site in the first column, the queue name in the second column, and the percentage of correct predictions in the remaining 5 columns.

In Table 3 we show a similar comparison of the RMS overprediction errors. The numbers represent the reciprocals of these numbers, normalized so that the value for "plain" is always 1, so that larger numbers in the table reflect predictions that are more accurate in our sense. That is, each number is the ratio of the "plain" RMS to that of the RMS denoted by the header of each column. In each row, we boldface the largest ratio among the correct predictions and denote with an asterisk the cases where the largest ratio comes from an incorrect predictor. From these two tables it is evident that, in general, BMBP using SCHMIB to cluster jobs by requested execution time is the most effective method.

4.4 Accuracy Analysis

As the results in the previous subsection indicate, clustering jobs by requested run time or by the product of run time and node count (node-seconds) tends to yield the most accurate bounds. In all but four of the cases (Datastar *high*, LLNL *all*, Lonestar *normal*, and SDSCBlue *low*) either the requested time or node-seconds SCHMIB method produces results which are correct (see Table 2) and have the largest RMS ratio (see Table 3). We suspect that the effect is a result of backfilling, in which the batch scheduler opportunistically runs short jobs while it is waiting for enough resources to become idle to run longer jobs. Since each job can be terminated when its maximum requested run time expires, the scheduler can set a deadline for a large job and then "backfill" the resources with jobs that it will terminate before the deadline. Anecdotally, in our experience, users generally believe that short jobs will experience less delay than long jobs will at the sites where we are currently running BMBP.

Somewhat more surprisingly, clustering by requested node count (either using SCHMIB or according to the static partitioning we discussed previously) is less effective. Most of the administrators who we have interviewed indicated that jobs requested fewer nodes would also enjoy preferential treatment, with the exception of the SDSC Datastar machine, leading us to suspect that node-seconds would yield the best results. At SDSC, the scheduling policy for Datastar has been to encourage large (in terms of node count) jobs, resulting in a complex interaction between the priority given to large jobs on the one hand and the ability to backfill

Machine/Queue	plain	static	nodes	rtime	rnprod
ctc/all	1.000	1.292	1.269	1.248	**1.615**
datastar/express	1.000	1.049	1.040	1.171	**1.262**
datastar/high*	**1.000**	0.928	0.960	1.725	1.333
datastar/normal	1.000	0.912	0.922	**1.190**	1.110
datastar/TGnormal	1.000	0.992	0.990	**1.317**	1.295
llnl/all	**1.000**	0.894	0.856	0.822	0.856
lonestar/normal	1.000	1.216	**1.793**	1.287	1.112
lonestar/serial	1.000	1.000	1.000	**1.074**	**1.074**
sdscblue/express*	1.000	0.946	0.965	**0.946**	0.819
sdscblue/high	1.000	1.079	1.042	**1.194**	1.046
sdscblue/low	**1.000**	0.944	0.883	0.909	0.824
sdscblue/normal	1.000	0.932	1.040	**1.202**	1.132
sdsc/express	1.000	1.113	1.037	1.131	**1.135**
sdsc/high	1.000	1.242	1.837	1.512	**2.899**
sdsc/low	1.000	1.350	1.273	**1.369**	1.287
sdsc/normal	1.000	1.006	0.979	**1.451**	1.127
ucteragrid/dque	1.000	1.058	1.068	**1.239**	1.100

Table 3. Root Mean Square (RMS) ratios for five BMBP simulation methods comparing the various grouping methods (static and SCHMIB) to a simulation run without grouping.

small jobs on the other. Because SCHMIB yields a higher RMS ratio when requested time or node-seconds are used than for node count, we conclude that these administrator-imposed policies are being overshadowed by the effects of opportunistic scheduling. Indeed, it may be that by artificially boosting the priority of large jobs, users are incentivized to submit even smaller jobs to ensure the effects of backfilling will be realized. We plan to explore further the use of SCHMIB as a diagnostic and analysis tool as part of our future work.

Also surprisingly, the administrator-determined static partitioning is never the most successful approach. In one case (Lonestar *normal*), SCHMIB is able to determine node clusterings that yield the largest RMS ratio, but in general clustering by nodes is a less desirable strategy. We were quite surprised by this result, since the administrators with whom we spoke were largely responsible for setting the scheduling policies for their respective machines. We note that SCHMIB (which adjusts its clustering dynamically) often achieves a more accurate result than the static approach leading us to suspect that user demands vary too dynamically for a static partitioning to be effective.

Finally, "plain" BMBP without SCHMIB is the most accurate approach in three cases. We discuss the Datastar *high* case in the next Subsection. In the case of the the LLNL queue, it appears from the log that all jobs go into a single queue that serves multiple machines of varying architectures and configurations. In such a heterogeneous node pool, it is possible that demand for a particular type of machine or machine configuration is the most important explanatory variable, which may even confound the variables we have considered. Finally, we had hypothesized before running these experiments that the *low* queues for both the SDSC SP-2 and the Blue Horizon would be most accurately forecasted by BMBP

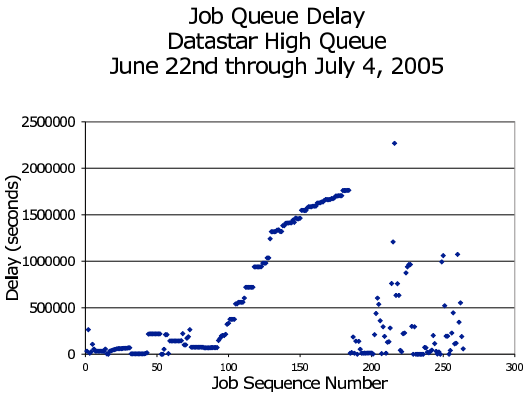

Figure 3. Queue delay measured in seconds, all jobs, Datastar *high* queue, June 22^{nd} through July 4^{th} 2005

alone. Our theory (which is refuted by the SDSC *low* ratio) was that the low queue was used for low-priority jobs and thus did not employ preferential treatment by job characteristics.

4.5 Correctness Analysis

Table 2 shows that SCHMIB fails to be correct in one case where "plain" BMBP succeeds: the Datastar *high* queue. As we mentioned previously, long runs of increasing delays will foil BMBP. In Figure 3 we show an 12-day-long period (June 22^{nd} through July 4^{th} 2005) from the Datastar *high* queue during which time such a spike in delay is evident.

The delay experienced by job #43 in this trace (submitted at 9:56 AM on June 24^{th}) was $15,840$ seconds (7 hours and 19 minutes). Job #44, submitted at 11:55 AM, experienced a queuing delay of $217,878$ seconds (60 hours and 31 minutes). From that point forward until job #184, submitted on June 27^{th} at 11:31 AM, the queuing delay increases monotonically for each successive job, peaking at $1,763,936$ seconds (20 days, 8 hours, 59 minutes). Thus, during a three-day period for this queue, the delay experienced by submitted jobs climbs steadily through three orders of magnitude. Note that the x-axis of the graph shows job delay in the order of submission sequence and does not depict the interarrival time between jobs. If it did, the monotonic trend would appear as an almost vertical "spike." Curiously, job #185, submitted on June 27^{th} at 11:55 AM, experienced a delay of only $11,359$ seconds (3 hours and 9 minutes) and thus began executing 20 days and 5 hours *sooner* than job #184.

Spikes such as the one depicted in Figure 3 occur on four separate occasions in the approximately 1.5 year-long trace. The *high* queue, we believe, is intended for higher priority jobs (which should experience lower queuing delays at a greater allocation expense) than jobs submitted to the other Datastar queues. As a result, we believe, only 6781 jobs were submitted during this period to the *high* queue, compared to $63,751$ in the *normal* queue.

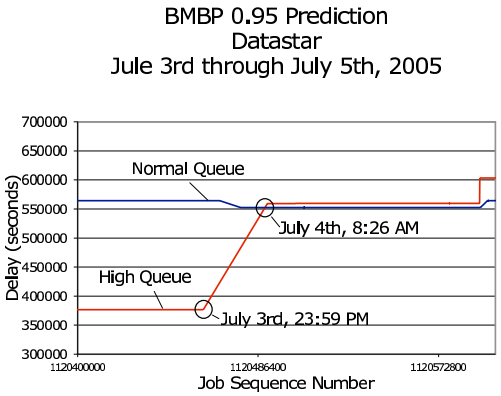

Figure 4. BMBP 0.95 predictions without SCHMIB, all jobs, Datastar *normal* and *high* queues, July 3^{nd} through July 5^{th} 2005

The size of the spikes combined with the relative infrequency of job submissions conspire to degrade the effectiveness of SCHMIB. When BMBP is used without SCHMIB, the history used for each prediction considers all jobs in the queue. When the first large spike in delay occurs, BMBP makes a series of incorrect predictions. However, the sparsity of the data causes it to include this spike in the valid history until the next similar episode. The intervening predictions are conservative but correct and when the next spike occurs, there are enough residual large values form the last spike to allow BMBP to make a sufficient number of correct predictions to ensure overall correctness.

When SCHMIB is added, the large values from the previous spike are present in, at most, the clustering corresponding to the longest wait times. Yet all jobs (regardless of their characteristics) appear to experience increasing delay. Thus, at best, SCHMIB would cause BMBP to return a correct prediction only for the cluster of largest values and job categorized into other clusters except for this largest one will see incorrect predictions. During the intervening periods the predictions are far less conservative, but ultimately SCHMIB degrades BMBP to the point where it does not achieve its correctness goal.

While this example demonstrates an uncontrived setting in which SCHMIB hinders BMBP, we believe it also illustrates the utility of both approaches (used in combination) as a diagnostic tool for the user or administrator.

In Figure 4 we show the 0.95 BMBP predictions without SCHMIB for both the Datastar *high* queue and the *normal* queue from July 3^{rd} through July 5^{th}. Starting late on the 3^{rd}, the predictions for the *high* queue start to increase as BMBP recognizes a series of consecutive misses as a change-point. By 8:30 AM on the 4^{th}, the predictions exceed those for the *normal* queue which, if nothing else, might have indicated a less-than-happy user community. We do not know whether this event corresponds to a policy or software failure, but if it does, the combination of BMBP's successful predictions without SCHMIB and the failure of SCHMIB to correctly categorize jobs seems to indicate a (possibly planned) priority inversion.

Interestingly, SCHMIB can also have the reverse effect and improve correctness, as indicated by the SDSCBlue *express* queue entry in Table 2. For this machine and queue, "plain" BMBP achieves a correctness percentage of 95.4% until very near the end of the trace, where a single spike causes a series of failed predictions. Because the failures occur so near the end, the eventual overall success percentage is below 95% (we analyze this phenomenon more completely in [26]). When SCHMIB clusters by requested execution time or node-seconds, however, the overall prediction is correct, despite the presence of a spike in delay near the end of the trace. In this case the data shows a series of spikes in rapid succession, with the largest occurring near the end. Because of the large variance in the data, SCHMIB maintains a conservative BMBP prediction in all clusters, so that the incorrect predictions near the end cause the correctness percentage for only the largest cluster to degrade below 95%. Because the other clusters achieve a sufficiently high success percentage, however, the overall predictor is correct.

Perhaps more generally, the data shown in Table 3 calls into question the way in which accuracy should be measured. Note that "plain" BMBP, even in cases where it achieves a lower RMS, mis-predicts jobs with relatively long delays. In other words, it will disproportionately mis-predict those jobs which would fall in the clusters corresponding to longer wait times. If, as we posit, these jobs also have the largest resource requirements, it is likely that the mis-predictions are for users who are most interested in accurate bounds. Conversely, SCHMIB attempts to "spread" the mis-predictions among various clusters evenly. As a result, even if it achieves a higher RMS error overall, users may find the results more useful.

4.6 Choosing a "Best" Strategy

While no single methodology is uniformly best, in the absence of specific information about the policies in place, using SCHMIB to cluster by requested execution time appears to be the most successful approach. Of the 13 cases where "rtime" or "rnprod" are the most accurate, in only two (CTC *all* and SDSC *high*) is "rnprod" better than 10% more accurate than "rtime." It is quite possible that these cases reflect some explicit node-hour-based administrative policy, especially in the case of the SDSC *high* queue, for which "rnprod" does so much better than any other method. We similarly postulate that the cases for which SCHMIB is less effective than "plain" prediction also reflect some policy (*e.g.*, lack of backfilling) which tends to homogenize the jobs' wait times.

However, it also may be possible to use an ensemble of SCHMIBs (one for each job characteristic) and simply to choose the one that has the lowers RMS over time. Using a 1-gigahertz Pentium III, the average time required to make each prediction over all batch queue logs is approximately 8 milliseconds. With this performance cost, it should be possible to run all methods simultaneously for each queue and then to track the RMS so that the most accurate can be selected in each setting.

5. Conclusions

Space-shared parallel computers use queuing systems for scheduling parallel jobs to processor partitions in such a way that each job runs exclusively on the processors it is given. In previous work [26, 26] we have proposed a method for estimating bounds on the queuing delay experienced by each each job and show that this non-parametric method – termed the Binomial Method Batch Predictor (BMBP) – outperforms competitive approaches.

Site administrators, however, use scheduling policies such as backfilling, as well as various explicit priority schemes, to balance the need for high resource utilization with the desire to minimize the queuing delay experienced by users. Many of these policies use requested execution time, requested node count, or the product of the two to determine how a job will be prioritized. Thus, taking these characteristics into account as a way of categorizing jobs should yield tighter predictions of queue delay bounds.

We therefore introduce SCHMIB, a hierarchical clustering scheme, to improve the accuracy of the bounds generated by BMBP, a non-parametric approach to estimating bounds for batch-queue wait times. We find that using SCHMIB to cluster jobs according to their requested time generally results in better predictions than those generated by BMBP alone and is more efficacious than clustering either by requested number of nodes or by requested number of node-seconds.

In the future, we will consider other approaches to the problem of clustering. No clustering method short of exhaustive search is guaranteed to find an optimal clustering (in terms of, in our case, BIC); it may be that for data of the type we are considering, we are better off using a partitional approach or model-based k-means. In addition, the sensitivity of clustering to chosen model should be investigated: it may be that by choosing a family of models that are more accurate than the exponential at the cost of some extra complexity (*e.g.*, Weibull distributions) will produce clusters for which BMBP can perform better. In addition, it is possible to employ a multidimensional clustering scheme, so that we can evaluate the effectiveness of using both requested time and requested nodes as clustering variables. There are also other variables (queue length, longest current wait time in the queue, requested processor type, requested memory, *etc.*) that might also be used as explanatory variables for wait times and could therefore be used as a basis for clustering. Finally, we plan to monitor the effects of BMBP and SCHMIB deployment for scientific user communities to discern its ultimate effects on workload and queuing performance.

6. REFERENCES

[1] IBM LoadLeveler User's Guide. Technical report, International Business Machines Corporation, 1993.

[2] G. Box, G. Jenkins, and G. Reinsel. *Time Series Analysis, Forecasting, and Control, 3rd edition*. Prentice Hall, 1994.

[3] S.-H. Chiang and M. K. Vernon. *Dynamic vs. Static Quantum-based Processor Allocation*. Springer-Verlag, 1996.

[4] S. Clearwater and S. Kleban. Heavy-tailed distributions in supercomputer jobs. Technical Report SAND2002-2378C, Sandia National Labs, 2002.

[5] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the 11th International Parallel Processing Symposium*, April 1997.

[6] A. Downey. Using queue time predictions for processor allocation. In *Proceedings of the 3rd Workshop on Job Scheduling Strategies for Parallel Processing*, April 1997.

[7] The Dror Feitelson's Parallel Workload Page. `http://www.cs.huji.ac.il/labs/parallel/workload`.

[8] D. G. Feitelson and B. Nitzberg. *Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860*. Springer-Verlag, 1996.

[9] D. G. Feitelson and L. Rudolph. *Parallel Job Scheduling: Issues and Approaches*. Springer-Verlag, 1995.

[10] D. G. Feitelson and L. Rudolph. *Towards Convergence in Job Schedulers for Parallel Supercomputers*. Springer-Verlag, 1996.

[11] E. Frachtenberg, D. G. Feitelson, J. Fernandez, and F. Petrini. *Parallel Job Scheduling Under Dynamic Workloads*. Springer-Verlag, 2003.

[12] C. Granger and P. Newbold. *Forecasting Economic Time Series*. Academic Press, 1986.

[13] Gridengine home page – `http://gridengine.sunsource.net/`.

[14] M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, June 1999.

[15] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[16] D. Lifka. *The ANL/IBM SP scheduling system*, volume 949. Springer-Verlag, 1995.

[17] D. Lifka, M. Henderson, and K. Rayl. Users guide to the argonne SP scheduling system. Technical Report TM-201, Argonne National Laboratory, Mathematics and Computer Science Division, May 1995.

[18] J. MacQueen. Some methods for classification and analysis of multivariate observations. pages 281–297, 1967.

[19] Maui scheduler home page – `http://www.clusterresources.com/products/maui/`.

[20] Cray NQE User's Guide – `http://docs.cray.com/books/2148_3.3/html-2148_3.3`.

[21] Pbspro home page – `http://www.altair.com/software/pbspro.htm`.

[22] C. Posse. Hierarchical model-based clustering for large datasets. *Journal of Computational and Graphical Statistics*, 10(3):464–??, 2001.

[23] G. Schwartz. Estimating the dimension of a model. In *Ann. of Statistics*, pages 461–464, 1979.

[24] W. Smith, V. E. Taylor, and I. T. Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *IPPS/SPDP '99/JSSPP '99: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pages 202–219, London, UK, 1999. Springer-Verlag.

[25] S. Z. Zhong. Journal of machine learning research 4 (2003) 1001-1037 submitted 3/03; revised 7/03; published 11/03 a unified framework for model-based clustering.

[26] Reference removed for the purpose of blind submission.